Chapter Five: Controlling the Motor – Programming the PWM

1. A Quick Introduction to Motors

The CalBOT is going to move on treads. A pair of motors drives these treads. You may wonder why we don't make a walking robot? If you look up research literature, you will find that legged robotics is still an exotic research subject. The reasons for these are many and complicated. A good starting point is the legged robotics lab at MIT: www.ai.mit.edu. For the CalBOT, we will be satisfied to use just a motor! Figure 1 shows how a motor works.

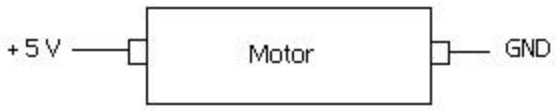

Figure 1. Motor turns in one direction.

You supply power to the motor as shown in figure 1 and the motor turns. If you want the motor turn in the other direction, simply reverse the connections. So, to drive the motor from your KitCON, you simple hook it up to one of the ports, and write a "1" or "0" as needed, correct?

WRONG! If you do this, you will end up blowing the board! The reason: the motor draws current when it is running. As a simple exercise, measure the current the motor draws by connecting your motor to the desktop power supply. You will see that it is much higher than the maximum current the board can supply (approximately 300 mA, refer to the KitCON hardware manual).

Even if you could drive the motor, you cannot control how fast your motor is turning. Your digital board can output only +5 V or 0 V, so you can just turn the motor on or off. Hmmm, so you want to accomplish three things:

1. Control the speed of the motor
2. Supply enough current for the motor
3. Change the direction of the motor so you can move backwards and turn

You control the speed of the motor using PWM, supply enough current for it by using the motor driver IC and make it move backwards or turn when the touch sensor is hit. The first two concepts are covered in this chapter; the touch sensor is the subject of chapter 6.

2. Pulse Width Modulation

First, some terminology associated with signals: frequency, period and duty cycle. The first important concept is the frequency of the signal, which is the number of cycles/second. The unit of frequency is cycles/sec or Hz.
The following signal has a frequency of 2 cycles/second or 2 Hz:

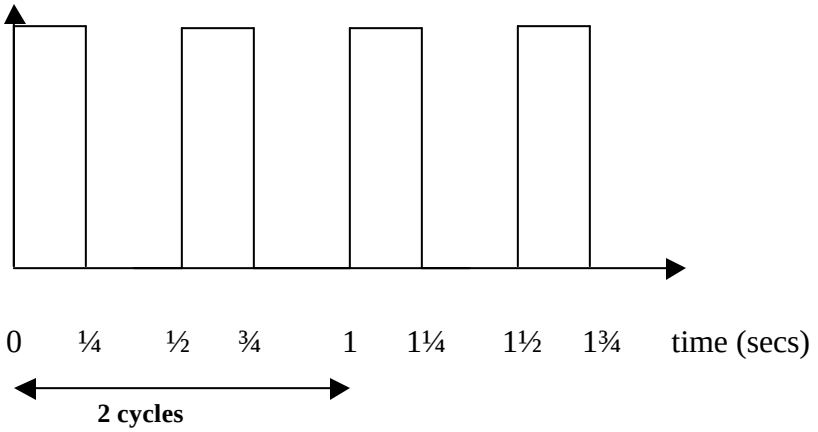

Figure 2. A signal with a frequency of 2 cycles/second or 2 Hz.

The next concept is the period. It is the reciprocal of frequency. It is simply the number of seconds to complete one cycle. The period of the signal in figure 2 is 0.5 seconds. Lastly, the duty cycle is defined as the percentage of the period for which the signal is "high". For example, a signal that has a 50% duty cycle is high for one half of the period and low for the other half. The signal in figure 2 has a 50% duty cycle.

(a) What is PWM?

As the name implies, a PWM(Reference 2) is a square wave in which we choose how long the signal stays logic high and how long the signal stays logic low. A PWM voltage waveform is illustrated in figure 3. The effective voltage delivered is just the peak voltage times the fraction of the time the voltage is on, that is, the average voltage. The greater this "on time", the higher the average voltage. The more voltage you supply to a motor, the faster it turns. Hence, if we can control the width of the time the signal stays logic high, we can control the speed of the motor.

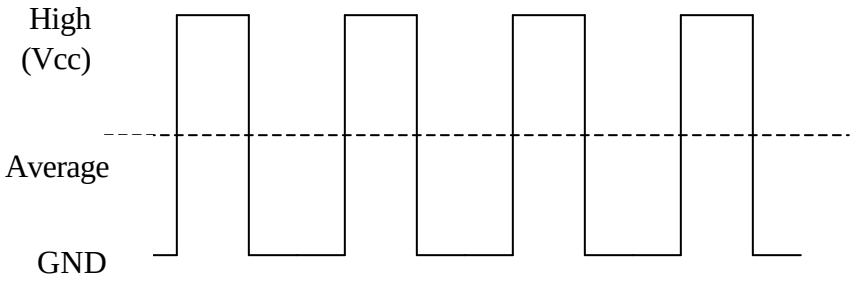

Figure 3. PWM signal. It has a duty cycle of about 40%, that is it is high for 40% the period.

(b) PWM using the C167[1]

As I mentioned in chapter 4, the C167 has two other types of registers (other than ports) to control a module. These two registers are the CONTROL REGISTERS and the DATA REGISTERS. Just like the ports of chapter 4, these registers also take on 16-bit integer values. However, they have special meaning to the C167. In the case of your C167's PWM module, you have the following:

Module Number	Control Register(s)	Data Register(s)
0	PWMCON0, PWMCON1	PP0, PW0
1	PWMCON0, PWMCON1	PP1, PW1
2	PWMCON0, PWMCON1	PP2, PW2
3	PWMCON0, PWMCON1	PP3, PW3

Table 1. The control and data registers for the PWM module

If you look at the section on the PWM in the hardware manual, it seems complicated. However, the most important concept to understand is how the control and data registers in table 1 work together to produce a square wave. You don't have to worry about all four modules. For the purposes of building the CalBOT, all four PWM modules in the C167 function the same way. You will be using only two of the four, in this manual I use module 0 and 1. If you want to use modules 2 and 3 or any combination of two out of the four, just use the correct data registers and set the control registers accordingly. Wait a second, what is this "setting the control registers" business? Well, I am going to describe the control and data registers in detail and this should clear things up. I am going to be describing module 0 only since all the other modules function the same way.

(c) Edge aligned PWM

The PWM module in the C167 can operate in one of four different modes: edge aligned, center aligned, single shot and burst mode. We will only be concerned with edge aligned PWM, refer to figure 4 for this mode, also called as mode 0. PLEASE NOTE: Appendix B also has figures 4 – 8 for easy reference.

[1] Throughout this section, refer to **The C167 User's Manual, pp. 15-1 to 15-13**.

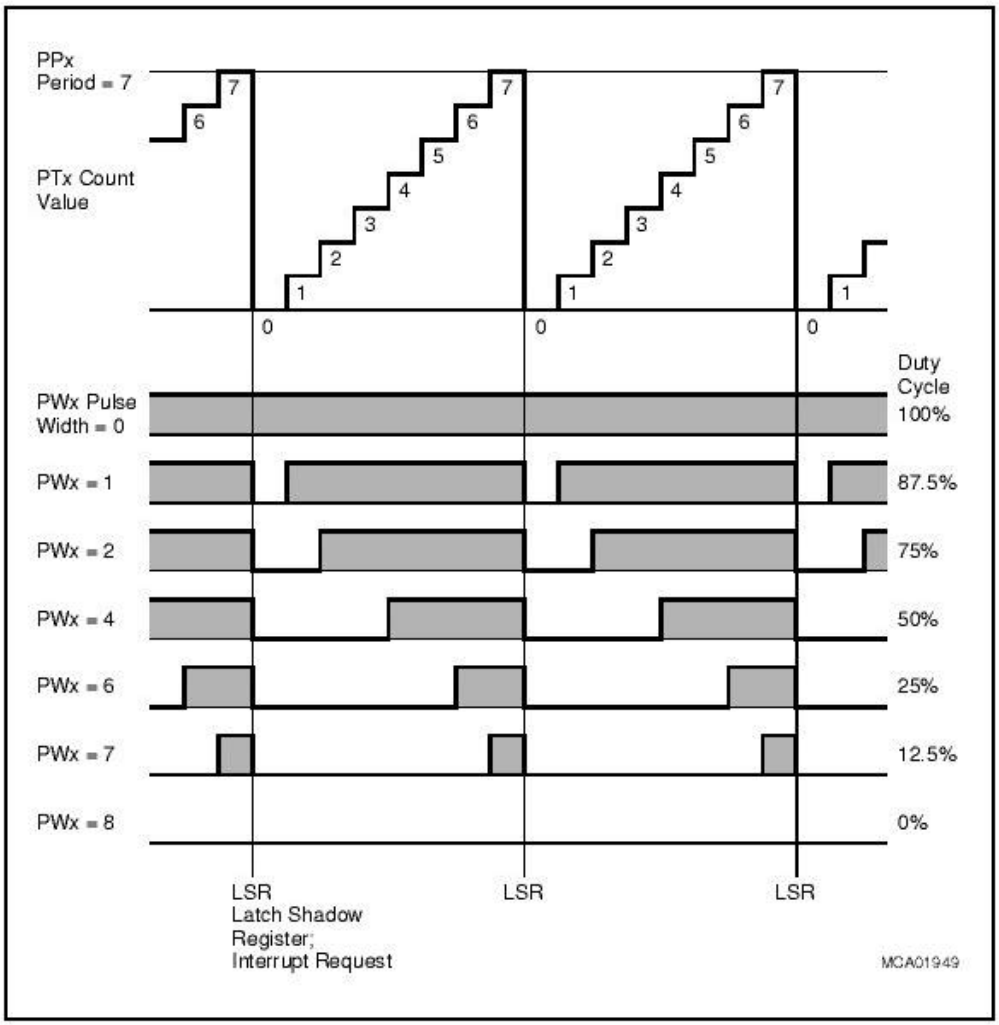

Figure 4. PWM Mode 0

Figure 4 shows what you have to do in order to get a square wave out of the PWM. We first set the period we want using PP0, *P*ulse *P*eriod register with the 0 standing for the module number, either 0, 1, 2 or 3. How do you specify the frequency you want? Figure 5 gives you the answer.

Inp.Clk. (f_{CPU}/x) (Counter resol.)	PWM Mode	8-bit PWM Resolution	10-bit PWM Resolution	12-bit PWM Resolution	14-bit PWM Resolution	16-bit PWM Resolution
20 MHz/1	0	78.13 KHz	19.53 KHz	4.88 KHz	1.22 KHz	305.2 Hz
(50 ns)	1	39.06 KHz	9.77 KHz	2.44 KHz	610.4 Hz	152.6 Hz
20 MHz/64	0	1.22 KHz	305.2 Hz	76.29 Hz	19.07 Hz	4.77 Hz
(3.2 µs)	1	610.4 Hz	152.6 Hz	38.15 Hz	9.54 Hz	2.38 Hz

Figure 5. Different PWM frequencies

(d) Specifying PWM Frequency

In figure 5, you can ignore the row under PWM Mode 1. As I said earlier, we will be using mode 0 only. In order to arrive at the frequency value, the C167 uses the onboard clock. The concept is extremely simple: we use the periodic clock signal to count up from 0. Since the on-board clock runs at 20 MHz, the period is 50 ns. In other words, we count once every 50 ns[2]. Once we reach the value we have in the PP0 register, we reset the counter and start again.

Suppose we have a value of 1023 in the PP0 register. Hence, the time it takes to count to 1023 is:

$$1023 \text{ counts} \times 50 \text{ ns/count} = 0.512 \ \mu s \qquad \text{Eq. (1)}$$

However, once we reach 1023, the counter is reset back to 0. So, we have a periodic reset every 0.512 μs. In other words, the *frequency* of the reset is:

$$1/0.512 \ \mu s = 19.53 \text{ KHz} \qquad \text{Eq. (2)}$$

Wait a minute, we see the value of 19.53 KHz in the first row under the column "10-bit PWM resolution" in figure 5. What does "10-bit PWM resolution mean"? Well, we set a decimal value of 1023 in PP0 to obtain 19.53 KHz. Since we are talking about *bit* resolution, what is 1023 in binary? You will see it is 1111111111, in other words it has 10 bits.

Hence, the formula for setting PP0 is:

$$PP0 = \text{floor}(1.0 / (\text{counter resolution (seconds)} \times \text{desired frequency (Hz)})) \qquad \text{Eq. (3)}$$

The bit resolution concept is not very important; I just mentioned it so that you won't be scratching your head when you read "10-bit PWM resolution". The table is divided into bit-resolutions just for convenience. The most important concepts are shown in Eq. (1) and Eq. (2). Thus, all you have to remember is that your period will depend on the product of your value in the PP0 register and your *counter resolution*. In our example, we said it takes 50 ns for a count. However, looking at figure 5, we see that we can make the C167 take 3.2 μs to count. These are the two possible counter resolutions. How do we tell the C167 which one to use? Well, we are going to be *controlling* counter resolution. See that word control? This should turn on some light bulbs in your head – control registers.

Whoa, hold on there. Before we go on to control registers, we should talk about how we specify the duty cycle.

[2] The C167 keeps track of the count in the PT0 register, see figure 4.

(e) Specifying PWM Duty cycle

Figure 4 again comes to our rescue. PW0, or *P*ulse *W*idth register 0 is used to specify the duty cycle. A value equal to (or greater than) PP0 results in a waveform that has a duty cycle of 0%, that is, the signal is always low. A value equal to approximately (PP0/2) gives a waveform of duty cycle 50%, high for one-half the period and low for the other. Hence, the duty cycle depends on the value in PW0 and is given by:

$$PW0 = (1 - (\text{required duty cycle}/100.0)) * PP0 \qquad \text{Eq. (4)}$$

The "required duty cycle" is a *float* [3] between 0 and 100. To summarize, the formulae for setting PP0 and PW0 are:

$$PP0 = \text{floor}(1.0 / (\text{counter resolution (seconds)} \times \text{desired frequency (Hz)})) \qquad \text{Eq. (5)}$$

$$PW0 = (1 - (\text{required duty cycle}/100.0)) * PP0 \qquad \text{Eq. (6)}$$

Please do not memorize the formulae above. Rather, try to understand how they came about. As a good exercise, try the following (in the order of increasing difficulty):

1. What is the PP0 value for a frequency of 45 KHz with a counter resolution of 50 ns?
2. What is the PP0 value for a frequency of 100 Hz?
3. What is the PP0 value for a frequency of 1 MHz?

The answers are given below, not in the same order as the questions. Fill in the correct question number in the parenthesis next to the answer. Figure 5, Eq. (5) and Eq. (6) are all you need for answering the questions. Once again, we are only concerned with mode 0 in figure 5.

( ) 3125
( ) 444
( ) 20

My solutions are explained on the next page.

[3] Although you could specify decimal values for the duty cycle, the value in PW0 is still an integer. So, you cannot get the exact decimal duty cycle you want.

Solution:

(2) 3125
(1) 444
(3) 20

1. What is the PP0 value for a frequency of 45 KHz with a counter resolution of 50 ns?

$$PP0 = 444$$

Plugging in the counter resolution and the frequency in Eq. (5) gives you the answer.

2. What is the PP0 value for a frequency of 100 Hz?

$$PP0 = 3125$$

Here we have not been specified the counter resolution. We can determine it from figure 5. In order to get a frequency of 100 Hz, you can't do it using the 50 ns counter resolution. The maximum integer that can be stored in PP0 is 65535 because this will take up all 16-bits. This corresponds to a frequency of 305.2 Hz only with the 50 ns counter resolution. Can we go lower than this? Yes, if we use the 3.2 μs counter resolution. Hence, using 3.2 μs and 100 Hz in Eq. (5), we get PP0 = 3125.

3. What is the PP0 value for a frequency of 1 MHz?

Look at figure 5. What is the highest frequency you can get? Remember PP0 can take on any value between 0 and 65535. A value of 0 in PP0 gives you a frequency of 0 in PP0. What about a value of 1 in PP0? Remember how the PWM works: the counter counts till the PP0 value, incrementing every counter resolution and resetting back to 0 when the counter reaches PP0. Well, if PP0 had a value of 1 then every 50 ns the counter would reset. This corresponds to a frequency of 20 MHz.

Hence, the maximum frequency you can get is 20 MHz (which is the frequency of your on-board clock). So, you should be able to do it with the 50 ns resolution. Plugging in 50 ns and 1 MHz in Eq. (5) gives you the answer. Incidentally, you can't get a frequency of 1 MHz using the 3.2 μs counter resolution. You can only go till a frequency of 3.125 KHz, corresponding to a value of 1 in PP0 when the counter resolution is 3.2 μs.

All we have left to do is to understand how to control the counter resolution, start the PWM and send it to the outside world. Here is where the control registers come in.

(f) PWM Control Registers

PWMCON0
PWM Control Register 0 **SFR ($FF30_H/98_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIR 3	PIR 2	PIR 1	PIR 0	PIE 3	PIE 2	PIE 1	PIE 0	PTI 3	PTI 2	PTI 1	PTI 0	PTR 3	PTR 2	PTR 1	PTR 0
rwh	rwh	rwh	rwh	rw	rw	rw	rw	rw	rw	rw	rw	rwh	rwh	rw	rw

Bit	Function
PTRx	**PWM Timer x Run Control Bit** 0: Timer PTx is disconnected from its input clock 1: Timer PTx is running
PTIx	**PWM Timer x Input Clock Selection** 0: Timer PTx clocked with CLK_{CPU} 1: Timer PTx clocked with $CLK_{CPU}/64$
PIEx	**PWM Channel x Interrupt Enable Flag** 0: Interrupt from channel x disabled 1: Interrupt from channel x enabled
PIRx	**PWM Channel x Interrupt Request Flag** 0: No interrupt request from channel x 1: Channel x interrupt pending (must be reset via software)

Figure 6. PWM control register 0.

PWMCON1
PWM Control Register 1 **SFR ($FF32_H/99_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS3	PS2	-	PB 01	-	-	-	-	PM3	PM2	PM1	PM0	PEN 3	PEN 2	PEN 1	PEN 0
rw	rw	-	rw	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
PENx	**PWM Channel x Output Enable Bit** 0: Channel x output signal disabled, generate interrupt only 1: Channel x output signal enabled
PMx	**PWM Channel x Mode Control Bit** 0: Channel x operates in mode 0, i.e. edge aligned PWM 1: Channel x operates in mode 1, i.e. center aligned PWM
PB01	**PWM Channel 0/1 Burst Mode Control Bit** 0: Channel 0 and channel 1 work independently in their respective standard mode 1: Outputs of channels 0 and 1 are ANDed to POUT0 in burst mode
PSx	**PWM Channel x Single Shot Mode Control Bit** 0: Channel x works in respective standard mode 1: Channel x operates in single shot mode

Figure 7. PWM control register 1

Figures 6 and 7 show the two PWM control registers. They are extremely simple to understand. First we enable the output and choose the edge-aligned PWM mode for the PWM module we want. I am using modules 0 and 1, so I set PWMCON1 accordingly:

PWMCON1 = 0x0003;

Next, we choose the counter resolution and start the timer. Suppose I want PWM module 0 to have a counter resolution of 3.2 µs and PWM module 1 to have a counter resolution of 3.2 µs. The value in PWMCON0 is:

PWMCON0 = 0x0033;

But, where does the output go to? If you read the first page of the PWM module chapter in the C167 hardware manual, you will see PWM0 through PWM3 are mapped to PORT7_0 through PORT7_3. Hence the PWM outputs go to port 7 in the C167. Thus, we need to set DP7_0 and DP7_1 to a 1.

```
DP7 = 0x03;    // Use P7_0 and P7_1 as output
P7  = 0x00;    // Initialize P7
```

PORT7_0 and PORT7_1 can be accessed using the KitCON-connector. Hence, all we have to do now is connect a logic analyzer input to PORT7_0 and PORT7_1. We can then see the PWM waveforms.

3. PWM Code

The code for turning on PWM module 0 and 1 and setting the correct frequency (100 Hz) for motor operation is shown below. I have added the function InitPWM() to our program from chapter 4 to correctly setup PWM module 0 and 1.

```
/* Header files */
#include <reg167.h>

/* Global variables and defines */


/* Initialization functions */

void InitLEDs(void) {
      DP2 = 0xFFFF; /* Turns on all LEDs */
      P2 = 0x0000;
}
```

```
/***************** ADDED IN CHAPTER 5 ***************/

void InitPWM(void) {
      DP7 = 0x03; /* Use P7_0 and P7_1 as output */
      P7 = 0x00;
      PWMCON1 = 0x0003;

      PP0 = 3125;   /* gives us a frequency of approximately 100 Hz */
      PW0 = 0.32*PP0;    /* 68% duty cycle */

      PP1 = 3125;
      PW1 = 0.50 * PP1;  /* 50% duty cycle */

      PWMCON0 = 0x0033;  /* choose correct resolution and start PWM */
}
/*******************************************************/

void Initialize(void) {
      InitLEDs();
      InitPWM();  /**** ADDED IN CHAPTER 5 ****/

      IEN = 1;    /* Enable Interrupts */
}

/* Utility functions */

void kludgyWait(void) {
      float i;

      /* Approx. 1/2 sec delay */
      for(i = 0;i < 2500;i+=0.1) {
            ;
      }
}

void main(void) {
      Initialize();

      for(;;)
            ;
}
```

Program 1. PWM code

Compile the above program in Keil, download it to the board and run it. Figure 8 shows one of the PWM signals on a portable oscilloscope. Please make sure you can get a PWM signal before proceeding to the next section, where we interface the board to the motors.

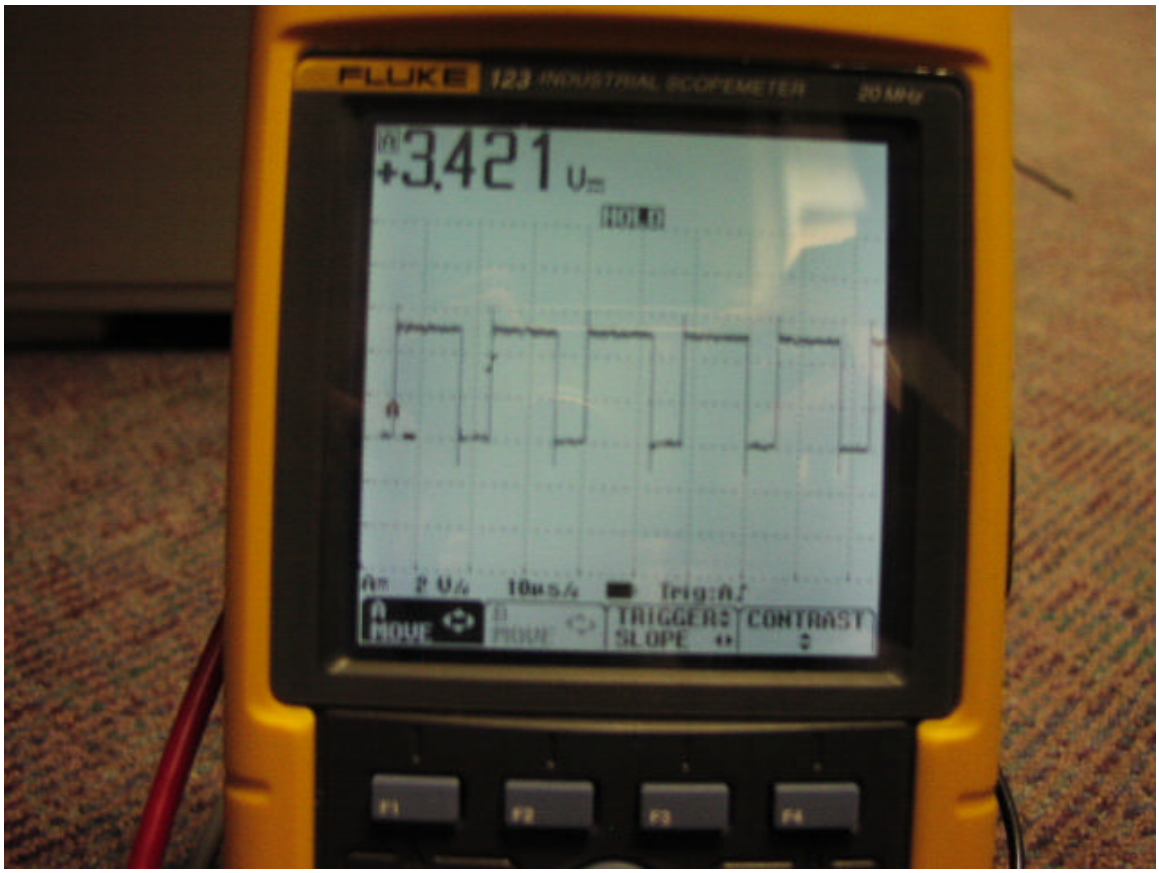

Figure 8. Scope output. The +3.421 V on the top left is the trigger level.

4. Supplying enough current - The SN754410

In order to supply enough current, you will be using a chip called a *motor driver*. The part number for this chip is the SN754410. It is manufactured by Texas Instruments. Figure 9 shows you how to mount this chip onto the KitCON-167.

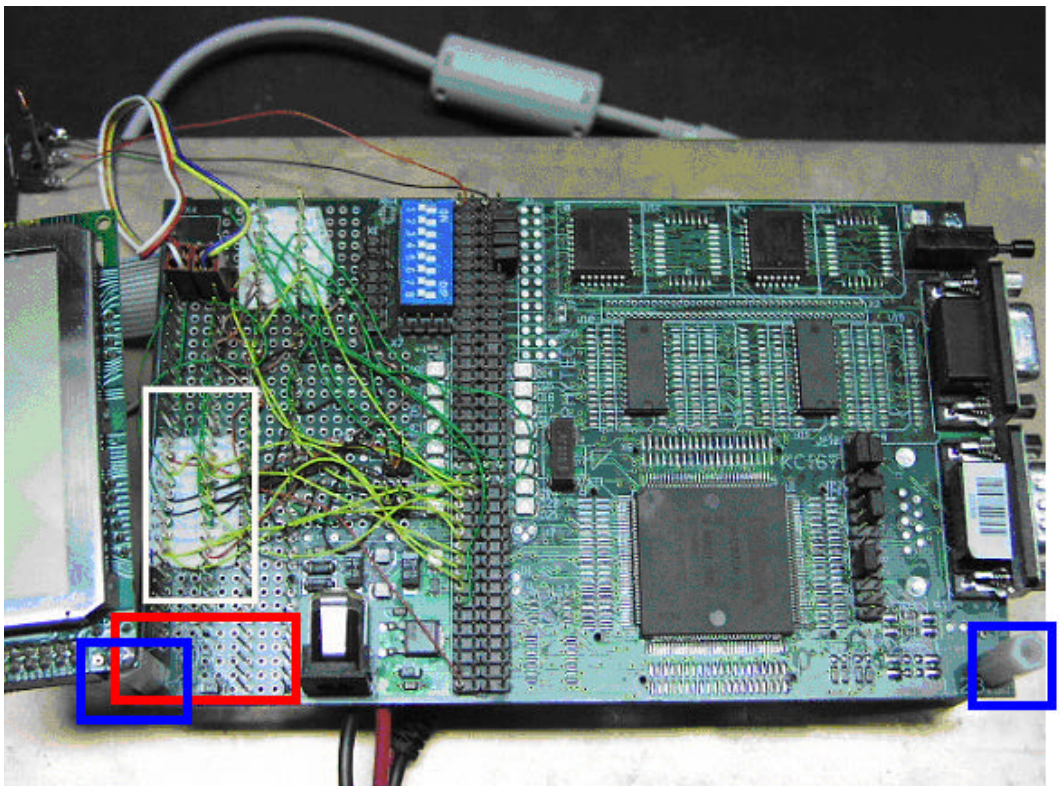

Figure 9. The white rectangle encloses my motor driver connections. You can unscrew the plastic standoffs enclosed by the blue rectangles and tilt the board up. You can insert the motor driver in the IC sockets underneath the KitCON. DO NOT insert the motor driver in the area enclosed by the red rectangle.

Figure 10 shows an approximate model of this chip and figure 11 shows the wire wrap connections that you need to make between this chip and the KitCON. Notice from figure 9 that I inserted a paper showing the different pins of the motor driver. This is always a good idea while wire wrapping. This extra work makes sure you don't get the pins mixed up.

Wire-wrapping the motor driver is probably the most tedious, boring and hardest part of the project. **PLEASE, PLEASE, PLEASE, PLEASE WIRE WRAP NEATLY AND CORRECTLY.** Doing so will save you a ton of time when debugging. Better yet, your project might require no debugging at all! Don't tell me I didn't warn you if the wire-wrap is bad! By the way, my wire wrap in figure 9 is horrible.

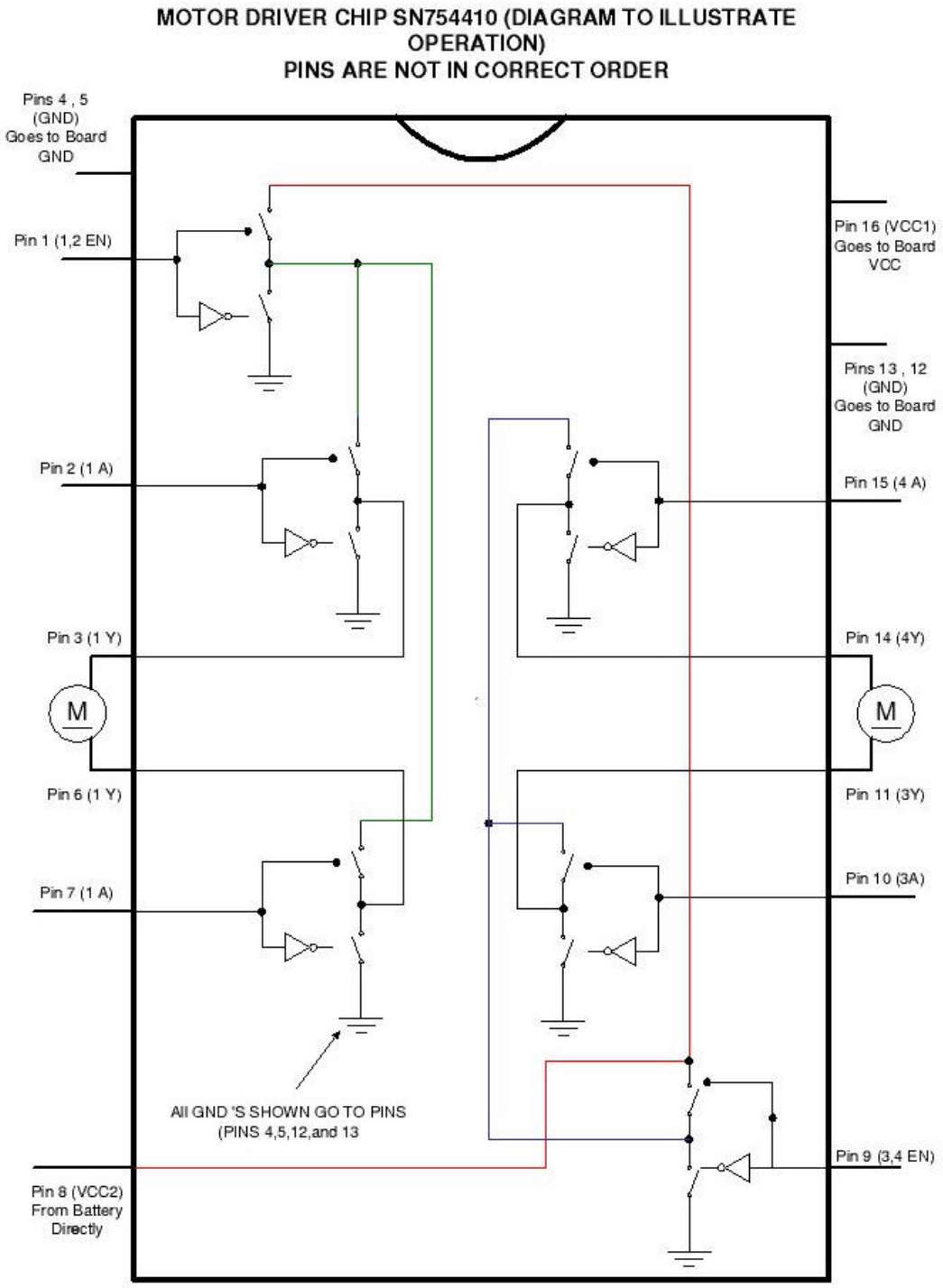
MOTOR DRIVER CHIP SN754410 (DIAGRAM TO ILLUSTRATE OPERATION)
PINS ARE NOT IN CORRECT ORDER
Pins 4 , 5 (GND) Goes to Board GND
Pin 1 (1,2 EN)
Pin 2 (1 A)
Pin 3 (1 Y)
M
Pin 6 (1 Y)
Pin 7 (1 A)
All GND 'S SHOWN GO TO PINS (PINS 4,5,12,and 13
Pin 8 (VCC2) From Battery Directly
Pin 16 (VCC1) Goes to Board VCC
Pins 13 , 12 (GND) Goes to Board GND
Pin 15 (4 A)
Pin 14 (4Y)
M
Pin 11 (3Y)
Pin 10 (3A)
Pin 9 (3,4 EN)

Figure 10. The motor driver dissected

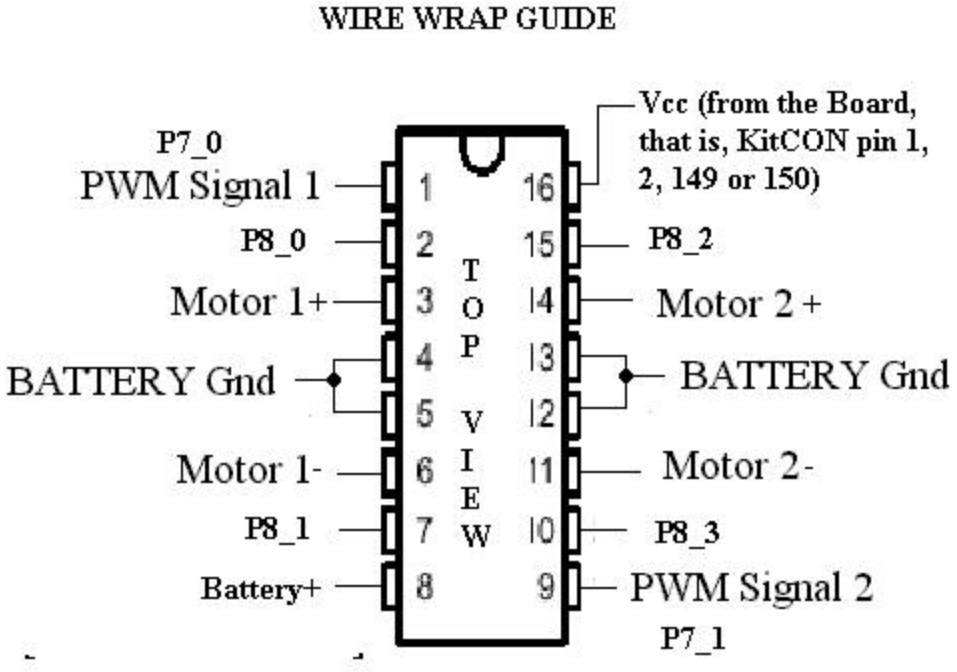

Figure 11. Wire wrap guide. P7_0, P7_1, P8_0, P8_1, P8_2 and P8_3 refer to PORT7_0, PORT7_1, PORT8_0, PORT8_1, PORT8_2 and PORT8_3 respectively.

We can understand how the motor driver works using figure 12. Figure 12 is just figure 11 zoomed in on the left hand side, along with port numbers from figure 11.

By the way, the four ground pins are connected to the die heat sink plate and are meant to be thermal grounds, but only one electrical ground is needed. Actually, it is better to connect this to the battery ground, not the kitcon ground[4].

[4] Thanks to Konrad Aeschanbach for this entry!

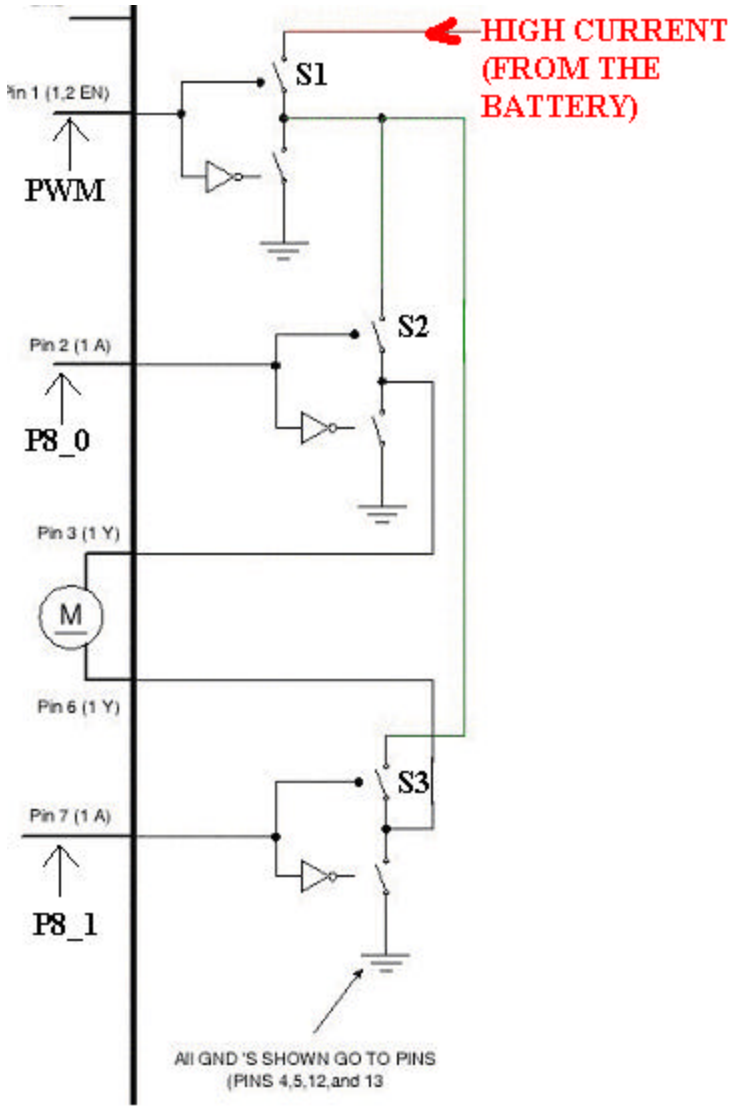

Figure 12. Motor driver operation

This side controls one motor; the motor is connected to pins 3 and 6 of the motor driver. Pins 2 and 7 of the motor driver control the direction of the motor. I have connected pins 2 and 7 to PORT8_0 (P8_0) and PORT8_1 (P8_1). We see pins 2 and 7 to be connected to two switches S2 and S3 respectively (don't worry about those inverters). These switches can be thought of as NMOS POWER FETs[5]. Since the gate of a FET is insulated, the current to run the motor has to come from the drain. Looking at figure 12 again, the drains of the FETs are connected through a common green wire to the drain of another FET, S1. The gate of this FET is the enable input and it is connected to the PWM. Hence, when the PWM[6] is high, this enable FET is on and the left-motor turns. Thus, if you want to turn the left motor:

[5] A POWER FET is a special FET that can handle large amounts of current to drive the motor. Nevertheless, the SN754410 does not use POWER FETs, it uses BJTs (Bipolar Junction Transistors). If you want to know more about them take EECS 105.

[6] DO NOT RUN YOUR MOTOR WITH PWM FREQUENCIES GREATER THAN 300 Hz.

1. Set the direction port for PORT8_0 and PORT8_1 to output:

DP8 = 0x03;

2. Write to PORT8_0 and PORT8_1 to turn the motor:

P8 = 0x02; /* 0x01 will make it go the other direction */

3. Turn on PWM connected to pin 1 of the motor driver.

Once the PWM goes low in the cycle, the enable FET goes off and the motor starts *coasting*. But, after a very short time the PWM turns on again and the motor turns on again. Hence, the motor sees the average voltage of the PWM. You can see how PWM controls the speed of the motor. Higher the duty cycle, the longer the PWM remains high, the higher the average voltage and faster the motor turns.

The most important observation about the motor driver is the difference between pins 8 and 16. Pin 8 drives the motor (refer to figure 11), so you should hook this up to the battery. Please, please be very cautious when connecting pins 8 and 16. **PLEASE DO NOT SWITCH THEM**. Doing so will destroy the board! If in doubt, **PLEASE ASK THE TA IN THE LAB**!

5. Motor code

The code to be added to the previous program in this chapter is shown below. Note that this assumes one motor is connected to PORT8_0 and PORT8_1 while the other is connected to PORT8_2 and PORT8_3. PLEASE NOTE: YOU ALSO NEED TO HAVE A DUTY CYCLE GREATER THAN 80% FOR THE MOTORS TO TURN. MODIFY YOUR PWM CODE ACCORDINGLY.

```
void main(void) {
      Initialize();

      /**** MOTOR CONTROL ****/
      DP8 = 0x03;  /* set direction */
      P8 = 0x02;   /* Turn the motor */
      /**** END MOTOR CONTROL ****/
      /* PWM CODE GOES HERE */
      for(;;)
            ;
}
```

6. Summary

This chapter showed you how to make the robot “walk”. As you read through this chapter, notice some subtle design problems you encountered:

1. Taking care not to blow the board – the motor driver.
2. Understanding PWM generation in the C167.

Just think about what you learned from this chapter: how PWM works in the C167. Before reading this chapter you probably had no clue as to what PWM is and how it is generated in the C167. This is common in engineering. 99% of the time, you will be asked to do something which you have no idea of about. What is the best way of doing such a thing? Well, your formal education is only a part of the solution. Most of the time, your experience and practical training will play the crucial role.

Before going on to connect the touch sensor to the CalBOT, it will be a good idea to make sure your motor driver works and the CalBOT at least moves forward. Remember the proverb: "You eat this elephant of life one bite at a time!"